Object Detection in E-Commerce Using YOLO in Real Time

Frans Mikael Sinaga[1*], Gunawan[2], Sunaryo Winardi[3], Heru Kurniawan[4], Wulan Sri Lestari[5], Karina Mannita Tarigan[6]

[1,2,3,4,5,6]Faculty of Informatics, Department of Informatics Engineering, Universitas Mikroskil, Medan, Sumatera Utara, Indonesia
Email: [1*]frans.sinaga@mikroskil.ac.id, [2]gunawan@mikroskil.ac.id, [3]sunaryo.winardi@mikroskil.ac.id, [4]heru.kurniawan@mikroskil.ac.id, [5]wulan.lestari@students.mikroskil.ac.id, [6]201111834@students.mikroskil.ac.id

(Received: 17 Feb 2024, revised: 14 Mar 2024, accepted: 15 Mar 2024)

Abstract

Presently, e-commerce platforms incorporate image search functionalities. Nevertheless, these systems possess constraints; input images necessitate static and manual cropping since the system does not automatically generate bounding boxes. Addressing this concern requires the implementation of an object detection algorithm to ascertain the quantity, location, and type of desired objects within real-time bounding boxes before users finalize their selection. This capability empowers users to readily discern their desired items, thereby augmenting the precision and efficiency of visual searches. Despite the availability of swifter object detection algorithms such as R-CNN and Mask R-CNN, which prioritize accuracy over speed, rendering them less suited for real-time detection, we opted to employ the YOLOv4 algorithm as an alternative, renowned for its efficacy in real-time object detection. Furthermore, we adopted the Color, Texture, and Edge-Based Image Retrieval (CTEBIR) technique for image matching. The results of our experimentation demonstrate that the utilization of the YOLOv4 algorithm can enhance the accuracy and speed of visual searches by streamlining the search process based on the identified classes. Additionally, our precision assessment yielded a score of 95%, with individual scores for camera objects reaching 90%, keyboards achieving 85%, and laptops attaining 71%. These findings corroborate the dependability of the CTEBIR algorithm in image matching and contribute to a deeper comprehension of the system's efficacy in accurately detecting and distinguishing objects.

Keywords: CTEBIR, Detection, E-commerce, Real-time, YOLOv4.

I. INTRODUCTION

E-commerce is a platform for buying and selling goods and services, as well as transferring funds or data over the internet, which is currently booming in the modern business world [1]. As a support system, e-commerce is equipped with search systems, whether text-based, audio-based, or image-based [2]. Image search has also become a common and important tool for studying many important topics ranging from spatial vision, attention, and oculomotor control to memory, decision-making, and rewards [3]. However, the visual search available in e-commerce platforms still has several weaknesses: users have to input a static image [2], and manual cropping of the object/item to be searched is required because bounding boxes are not automatically provided by the system [4]. Bounding boxes in image detection systems are useful for increasing detection speed and limiting the area of image checking for comparison with data in the system, thus improving matching accuracy with system data. Moreover, in image-based searches, an image can consist of multiple objects, so an object detection algorithm is needed to determine the quantity, location, and type (classification) of desired objects in the form of bounding boxes in real-time before users make a selection. In addition to helping users ensure the type of object they want to search for, the results of object detection can also enhance search accuracy and efficiency [5].

In detecting an image, various extraction methods such as color, texture, and edges can be employed. Color filtering is a specific image processing technique based on a particular color. The working principle of color filtering involves comparing the color components of each pixel in the image with a specific color [6]. However, relying solely on color features may not be sufficient for object detection due to the significant influence of lighting conditions and the presence of numerous items during real-time image search. Therefore, it is necessary to consider using other features for extraction, such as texture and edges. YOLOv4 is one of the object detection algorithms known for its excellent performance in terms of both accuracy and detection time [7]. YOLOv4 also boasts a simple architecture, trained to perform classification and bounding box regression simultaneously, making it fast and suitable for real-time detection [8]. Additionally, YOLOv4 is a cleaner object detection method due to its end-to-end training process [9].

As a solution, a real-time image search system will be developed, implementing the YOLOv4 object detection algorithm along with the Color, Texture, and Edge-Based Image Retrieval (CTEBIR) technique in a web-based e-commerce application. The proposed method will start by inputting a set of images in real-time, which will be processed by the YOLOv4 algorithm to predict a number of bounding boxes and only display boxes with confidence scores above 0.25. This step serves as an initial stage for detection before applying various proposed features. The YOLOv4 algorithm will also classify the type of object based on conditional class probabilities to narrow down the database's data coverage for comparison with the input image during the image retrieval stage [10], [11]. Next, users will be asked to select one object (bounding box), which will then be cropped by the system, proceeding to the image retrieval stage using the CTEBIR technique. In the initial stage, each image in the database is selected based on color similarity and taken to form a database subset. Then, the Local Binary Pattern (LBP) and Canny Edge Detection methods are used to extract texture and edge features from the query image and images in the subset from the first stage. Subsequently, the Manhattan distance information between two corresponding features of the query image and the selected image is computed and combined, then sorted using the bubble sort algorithm [12]. Finally, users receive a catalog sorted from products with the highest to lowest similarity values in their images [13]. The image with the highest similarity value will be the final result of the detection process.

The expected outcome of this research is to accurately detect images in real-time, thereby enhancing user comfort in utilizing image search features, expanding insights into object detection, and serving as a reference for other researchers in the field of object detection using CTEBIR features.

II. RESEARCH METHODOLOGY

A. Research Stages

Visual search is an intricate process encompassing vision, comprehension, memory, decision-making, and reward, fundamentally altering the dynamics of how individuals engage with surrounding products. The object of interest sought by an observer is termed the "target," whereas other items are designated as "distractors." Contemporary visual search technology leverages Artificial Intelligence (AI) to comprehend the content and context of images, presenting a curated list of pertinent results [1].

Rather than requiring humans to adopt a computational mindset, as in text-based searches, visual search analyzes images utilizing visual cues and image metadata. Facilitated by AI, visual search delivers the most pertinent results by identifying similarities, such as specific colors or styles. Undoubtedly, this facilitates a smoother retail experience, enabling customers to swiftly locate desired items [2].

B. Images

An image comprises distinct points that collectively form a cohesive entity imbued with meaning, encompassing both "artistic" and "intrinsic" aspects. A high-quality image not only showcases the aesthetic appeal of the composition (artistic) but also ensures clarity for analysis and various applications (intrinsic). Pixels, the smallest units composing an image, govern its resolution and contribute significantly to its overall quality and visual fidelity. represent the units of an image. Pixel is short for picture The "element" refers to the graphic representation of the smallest unit in a graphical image, measured per inch. Each pixel not only represents a single point in an image but also constitutes a component within a box, commonly known as a cell, which is the smallest unit of the image [1].

There are two main types of images: analog images and digital images. An analog image is continuous, such as a photograph captured with an analog camera or the display on a TV or monitor (video signal). In contrast, a digital image is stored on a storage medium and can be manipulated by a computer. Images can further be classified into two categories: still images and moving images. Still images are presented sequentially, creating the illusion of motion to the viewer. Each image in the sequence is referred to as a frame. Widescreen films or television shows typically comprise hundreds to thousands of frames [14].

C. Algorithm You Only Look Once (YOLO)

The YOLO (You Only Look Once) system is a real-time object detection system that can identify multiple objects simultaneously within a single frame. YOLO outperforms other object detection systems in terms of accuracy and speed. It can predict up to 9,000 classes, including classes that may not be visible. YOLO's primary objective is to locate specific objects within an image and classify them. In essence, it takes an image as input and produces a vector containing bounding boxes and class predictions.

The YOLO algorithm is a real-time object detection algorithm that prioritizes speed and recognition over spending excessive time on creating region proposals, thereby aiming for efficient object detection rather than achieving perfect detection of all objects [15]. The primary objective of researching this algorithm is to develop an object detector optimized for rapid operation in production systems and parallel computation, rather than focusing solely on low computational volume theoretical indicators (BFLOP). The YOLO algorithm is user-friendly and straightforward to train, enabling even conventional GPU users to achieve real-time, high-quality, and convincing object detection results [10]. The detection system operates by utilizing a repurposed classifier or localizer for detection purposes. The model is applied to an image across various locations and scales. Areas containing the highest scored images are identified as detections [10].

D. Research Methodology

The flowchart of the overall real-time visual search system analysis process in e-commerce can be seen in Figure 1.

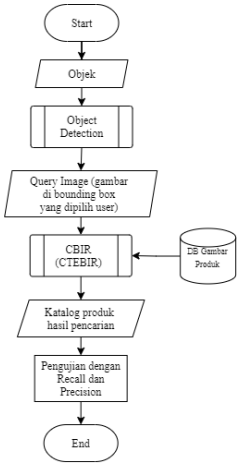

Figure 1. The Flowchart of The Entire Process of the Image Search System.

E. Input Object

The input objects consist of a series of images captured by the user's camera while scanning for the desired object in real-time.

F. Object Detection

Object detection is performed using the YOLO algorithm, which consists of 2 main processes: training and validation. The input for this process includes a set of images from stage 1 and a dataset. The output of this process includes bounding boxes and classifications of the recognized object classes by the YOLO model. The object detection process consists of:

1. Prepare the dataset

The dataset of electronic items is obtained from the OpenImagev6 dataset and downloaded using the OIDv6 application. The dataset consists of 16,760 images, with 15,512 images for training and 1,248 images for validation. It is divided into 6 different categories of electronic items: camera, printer, laptop, tablet, keyboard, and mouse, with specifications as shown in the following Table 1:

Table 1. Dataset Information

Category	Training	Validation
Camera	5037	484
Keyboard	3331	287
Mouse	622	100
Tablet	784	54
Printer	210	74
Laptop	5528	249

2. Preprocessing Image Data

Next, annotation preprocessing is performed from the OIDv6 format to the YOLO format as shown in Figure 2 and Figure 3.

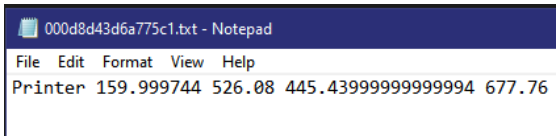

Figure 2. The OIDv6 Annotation Format (Before Conversion)

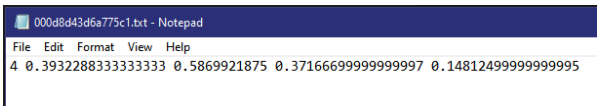

Figure 3. The YOLO Annotation Format (After Coversion)

3. Configuration of the YOLO Network

The network configuration is required as the model network to load the data for training. The number of images in one batch is 64, meaning that each iteration loads 64 images. Then, the batch is divided into several blocks that can run in parallel on the GPU. The number of subdivisions or batch blocks is set to 64 so that the modeling process divides the batch into 64 parts. Additionally, changes in learning rate are made after batch 10,000 and 11,250. The maximum batch processed in one iteration is 12,500. The input size is 576x576. The class value in the [yolo] layer is set to 6 according to the number of classes to be trained, and the filter value in each layer before the yolo layer is 33. All these parameter values are calculated based on the Darknet documentation.

4. Darknet Configuration

Darknet configuration is necessary to ensure the data training process runs smoothly. This configuration process begins by creating two files named obj.names and obj.data, which are stored in the data directory. The obj.names file contains the names/classes of the detected objects. Each class is separated by a new line.

Then, the obj.data file contains the configuration for training YOLO, including the number of classes, the location of the list of image files for training and validation in .txt file format, the names of object classes, and the backup directory used to store temporary training weight results. Temporary training weight results are usually saved if the model after validation and mAP value on that model is better than the previous validation. The contents of the configuration in the obj.data file can be seen in Figure 4 below.

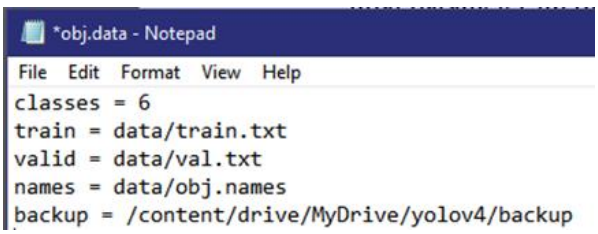

Figure 4. The Content of the Obj Data File

After creating the configuration files, the next step is to configure Darknet to run training with GPU. Once the building process is complete, the next step is to download the

pre-trained weights file for YOLOv4 and place it in the root directory of Darknet.

5. Training the YOLO Model

The YOLO Model Training is conducted by executing the command/darknet detector train data/obj.data cfg/YOLOv4-obj.cfg YOLOv4.conv.137 -map -dont_show where data/obj.data is the location of the Darknet configuration file, cfg/YOLOv4-obj.cfg is the configuration file of the YOLOv4 model, and YOLOv4.conv.137 is the pre-trained weight file for YOLOv4. -map is the argument to validate the mAP value every 4 epochs.

G. Query Image

The query image is an image within a bounding box chosen by the user among the objects recognized by the YOLOv4 model and automatically cropped by the system. The class resulting from object classification is also used to narrow down the search space.

H. CBIR / CTEBIR (Color, Texture, and Edge Based Image Retrieval)

CTEBIR is one of the methods in CBIR (Content-Based Image Retrieval). The algorithm works by initially selecting suitable images from a large database based on color moment information, which is then stored in a new database (subset). Next, the Local Binary Pattern (LBP) and Canny edge detection methods are used to extract texture and edge features from the query image and the images in the subset obtained from the first stage. Then, the Manhattan distance information between the corresponding features of the query image and the selected image is calculated and combined, and then sorted using the bubble sort algorithm. The workflow of the image processing process in performing detection can be seen in Figure 5 below.

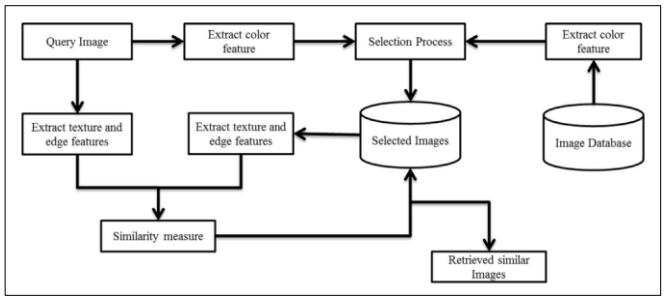

Figure 5. Block Diagram of Color, Texture, and Edge-based CBIR [7].

1. Color Descriptor

In this algorithm, to minimize complexity and enhance the effectiveness of CBIR, a global color descriptor is used at the first retrieval level. Color moments (statistical measures) are selected to represent the color details of the image. This provides information on the distribution of pixel colors in two forms of moments. The first-order moments provide average information about the pixel distribution of a specific image (Mean), and the closeness of the pixel distribution to the average color is estimated by the second-order moments (Standard Deviation). In the first stage of the retrieval process, the average color information (mean) and the quantity of different pixel values from the mean (standard deviation) of the query image are globally estimated from the three color channels (Red, Green, Blue) of the RGB color space using Equations (1) and (2). If the pixels in the image are close to the mean value, the standard deviation will be low. High standard deviation indicates that a large number of pixel colors do not approach the mean value.

$$Mean(Ic) = \frac{1}{M\ x\ N} \sum_{i=1}^{M} \sum_{j=1}^{N} Pcij\ , c = \{R, G, B\} \quad (1)$$

$$Std(Ic) = \left(\frac{1}{M\ x\ N} \sum_{i=1}^{M} \sum_{j=1}^{N} (Pcij - Mean(Ic)^2\right)^{\frac{1}{2}}\ , c = \{R, G, B\} \quad (2)$$

Where, Ic = color channel information of an image, M = size of rows, N = size of columns of an image, Pcij = pixel value of an image at row i and column j in a specific color channel.

Only the Mean (Ic) value is required to select images from the database to generate a smaller search space. It cannot be ensured that the Mean value is one of the accurate pixel information from a given specific image. The image standard deviation is important to provide details about the distribution of image pixels around the mean information. This information acts as lower and upper bounds for the mean value and allows taking values between them. Therefore, the details of the standard deviation are added and subtracted from the Mean value. The result of this operation provides two threshold values for each channel, namely: Low Threshold (LT) as in Equation (3) and High Threshold (HT) as in Equation (4) given as follows:

$$LT(Ic) = Mean(Ic) - Std(Ic), c = \{R, G, B\} \quad (3)$$

$$HT(Ic) = Mean(Ic) + Std(Ic), c = \{R, G, B\} \quad (4)$$

If the Mean (Ic) of an image from the database lies between the two threshold values (LT and HT), then that image is selected for the subsequent feature extraction process. The threshold values for the R, G, and B color channels are combined using the logical AND operator (&&). Here, the first stage of the proposed work behaves as a filter that takes all images from the database and passes the images that meet the specified criteria at this level. At the end of this first stage, the collectively selected images form a subset of the original database. The subsequent stages use this subset of images rather than the original database for image retrieval.

2. Texture Descriptor

Texture is another prominent descriptor in CBIR systems. This extraction algorithm is performed on the selected subset of images from the first stage of the retrieval process. Before exploring LBP on the selected images, an RGB to grayscale transformation is performed as a preprocessing step on those images. For each iteration, a 3 × 3 overlapping grayscale image is required as input. The pixel values available at the

Center Position (CP) of the 3 × 3 sub-block act as the threshold values for its neighboring pixels. Using these threshold values, a binary representation of the sub-block is created. Then, the LBP value of the 3x3 sub-block is evaluated in a clockwise direction. Finally, the LBP value is updated at the center pixel position of the block in the image. The first iteration of LBP is illustrated in Figure 6 (a) and Equation (5) to show the LBP estimation on the 3 × 3 block representation.

$$LBP_N(Ic) = \sum_{i=0}^{N-1} f\,(P_i - CP)2^i\,; f(p) = \begin{cases} 1; P \geq 0 \\ 0; P < 0 \end{cases} \quad (5)$$

Where N = total neighboring pixels for CP in the 3 × 3 sub-block.

In Figure 6 (b), the first 3 × 3 sub-block is taken from a simple grayscale image sized 5 × 5. In this case, the pixel value 21 at the center is the threshold value for its 8 neighbors. The difference between the value of each neighboring pixel and the value of the center pixel is calculated. If the difference value is greater than or equal to 0, then the pixel value is changed to 1; otherwise, it is updated to 0 instead. Next, this 8-bit binary value is converted to a decimal value and rehabilitated in place of the center pixel, as shown in Figure 6 (c). After obtaining the LBP for the entire image, the histogram of LBP values is calculated, which provides a representation of the texture features of an image.

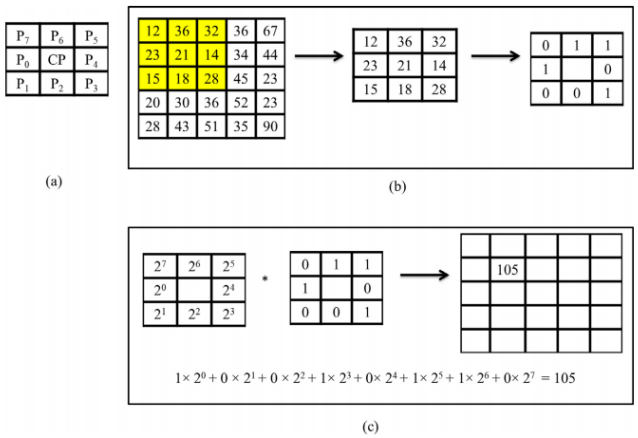

Figure 6. First Iteration of LBP: (a) Position of Pixels in the 3 × 3 Block Image (b) Generation of Binary Values Using Threshold Values (c) Generation of LBP Values Through the Obtained Binary Values [8].

3. Edge Descriptor

Usually, edges are formed by sudden changes in the intensity values of the image captured by the edge detection algorithm and hold the representation of the boundaries of objects present in the image. Canny edge detection is used to represent the shape of objects in the selected image at the end of the first stage. Initially, color edge features are separated into R, G, and B channels. The Canny Edge Detection algorithm consists of 5 steps [16], including:

a. Noise Reduction.

Since the mathematics involved behind the scenes are mostly based on derivatives, edge detection results are highly sensitive to image noise. One way to remove noise from the image is by applying Gaussian blur to smoothen it. To do this, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7, etc.). The kernel size depends on the blurring effect desired. Essentially, the smaller the kernel, the less noticeable the blur. The equation for the Gaussian filter kernel size (2k + 1) × (2k + 1) can be seen in Equation (6).

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right); 1 \leq i,j \leq (2k+1) \quad (6)$$

To begin, a two-dimensional Gaussian function is required as in Equation (7).

$$G\,(x,y) = \frac{1}{2\pi\sigma^2}^{-\frac{x^2+y^2}{2\sigma^2}} \quad (7)$$

The values of this function will create a convolution matrix/kernel that we will apply to each pixel in the original image. The kernel is usually quite small - the larger it is, the more computations we have to perform on each pixel. Here, x and y determine the delta from the central pixel (0, 0). For example, if the chosen kernel radius is 3, x and y will range from -3 to 3 (inclusive). The standard deviation - affects how significantly neighboring pixels influence the computation result of the central pixel.

b. Gradient Calculation.

The gradient calculation process detects the intensity and direction of edges by computing the image gradient using edge detection operators. These edges correspond to changes in pixel intensity. To detect them, the simplest way is to apply filters that highlight these intensity changes in both directions: horizontal (x) and vertical (y).

When the image is smoothed, the derivatives Ix and Iy with respect to x and y are computed. This can be implemented by convolving I with Sobel kernels Kx and Ky, respectively (Figure 7).

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Figure 7. Sobel Filter for Both Directions (Horizontal and Vertical)

The formula yields the magnitude matrix Equation (8):

$$magnitude\,(I) = \sqrt{x\,(I)^2 + y(I)^2} \quad (8)$$

The formula yields the angle matrix Equation (9):

$$angle(I) = atan2(y(I), x(I))[\cdot\, 180/\pi] \quad (9)$$

The result is almost as expected, but it can still be observed that some edges are thick while others are thin. The next step, Non-Maximum Suppression, will help reduce the thick lines. Additionally, the intensity levels of the gradient between 0 and 255 are not uniform. The edges in the final result should have the same intensity (e.g., white pixels = 255).

c. Non-Maximum Supression.

Ideally, the final image should have thin edges. Therefore, Non-Maximum Suppression (NMS) should be performed to thin out the edges. The principle used is to traverse all points in the gradient intensity matrix and find pixels with maximum values in the edge direction. The illustration of the traversal direction can be seen in Figure 8 below.

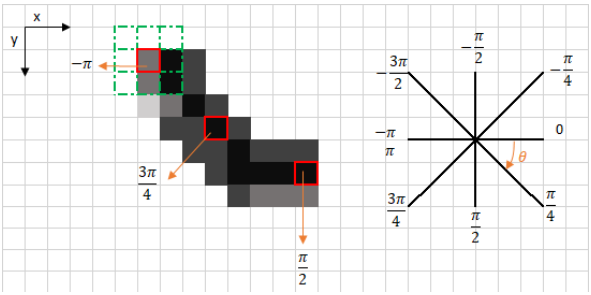

Figure 8. The Traversal Direction in NMS Corresponds to the Rotation of Angles.

The top left corner of the red box in the above image represents the intensity pixels from the Gradient Intensity matrix being processed. The corresponding edge direction is represented by the orange arrow with an angle of -π radians (+/- 180 degrees) as shown in Figure 9.

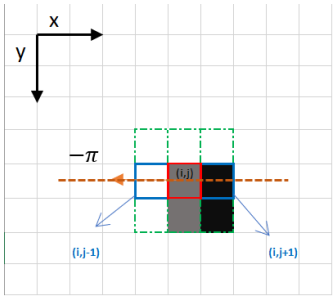

Figure 9. Checking Pixels in the -π Radian Direction

Edge directions are represented by the dashed orange lines (horizontal from left to right). The goal of this algorithm is to check whether pixels in the same direction are more or less intense than the one being processed. In the example above, pixel (i, j) is being processed, and pixels in the same direction are highlighted in blue (i, j-1) and (i, j+1). If either of these two pixels is more intense than the one being processed, only the more intense pixel is retained. Pixel (i, j-1) appears to be more intense because it is white (value 255). Therefore, the intensity value of the current pixel (i, j) is set to 0. If there are no pixels in the edge direction that have a stronger value, then the intensity of the current pixel is retained.

In this case, the direction is the diagonal line represented by the dashed orange dots. Therefore, the strongest pixel in this direction is pixel (i-1, j+1). So, each pixel has 2 main criteria (edge direction in radians, and pixel intensity (between 0–255)). Based on this input, the steps of non-maximum suppression are:

1. Create a matrix initialized to 0 with the same size as the original gradient intensity matrix;
2. Identify the edge direction based on the angle values from the angle matrix;
3. Check whether pixels in the same direction have higher intensity than the pixel being processed;
4. Return the processed image using the NMS algorithm.

d. Double Treshold

This step aims to identify 3 types of pixels:

1. Strong pixels are pixels with very high intensity values that are believed to contribute to the final edge.
2. Weak pixels are pixels with intensity values that are not strong enough to be considered strong, but not small enough to be considered irrelevant for edge detection.
3. Other pixels are considered irrelevant for edges.

e. Edge Tracking by Hysteresis.

Based on the thresholding results, hysteresis turns weak pixels into strong ones if and only if at least one pixel around the pixel being processed is a strong pixel.

4. Similarity Measure

The similarity measure is essential in all types of retrieval systems because it provides a measure of the distance between the low-level visual content of two images. The distance information is a determinant factor of the similarity measure. A very low resultant value indicates that the matching database image is very close to the given query image. LBP and edge feature similarity are estimated through the Manhattan distance measurement given by Equations (10) and (11).

$$LBP_{SM(QLBP_{IMAGE},NewDBLBP_{COUNTIMAGES})} = \sum_{i=1}^{N} \left| f_{QLBP_{IMAGE}}(i) - f_{NewDBLBP_{COUNTIMAGES}}(i) \right| \quad (10)$$

Where $f_{QLBPIMAGE}$ (i) is the i^{th} LBP feature of the query image, $f_{NewDBLBPCOUNT\ IMAGE}$ (i) is the i^{th} LBP feature of an image in the new database, and N is the total number of LBP features in the image.

$$EDGE_{SM(QEDGE_{IMAGE},NewDBEDGE_{COUNTIMAGES})} = \sum_{i=1}^{N} \left| f_{QEDGE_{R\ IMAGE}}(i) - f_{NewDBEDGE_{R\ COUNTIMAGES}}(i) \right| + \sum_{i=1}^{N} \left| f_{QEDGE_{G\ IMAGE}}(i) - f_{NewDBEDGE_{R\ COUNTIMAGES}}(i) \right| + \sum_{i=1}^{N} \left| f_{QEDGE_{B\ IMAGE}}(i) - f_{NewDBEDGE_{R\ COUNTIMAGES}}(i) \right| \quad (11)$$

Where $f_{QEDGE\_RIMAGE(i)}$, $f_{QEDGE\_GIMAGE\ (i)}$ dan $f_{QEDGE\_BIMAGE\ (i)}$ represent the i^{th} edge feature of the query image in the R, G,

and B color channels, respectively. Similarly, $f_{NewDBEDGE\_RCOUNTIMAGES(i)}$, $f_{NewDBEDGE\_GCOUNT\ IMAGE(i)}$ and $f_{NewDBEDGE\_BCOUNT\ IMAGE}$ provide the i^{th} edge feature of a new database image in the R, G, and B color channels, respectively. NR, NG, and NB correspondingly denote the number of edge features in the R, G, and B color channels of an image.

LBP and edge feature distance values obtained randomly vary from each other in an unbounded manner. Therefore, normalization is important to confine large and small variations in feature values to the range [0, 1]. Min-Max normalization is implemented through Equations (12) and (13) on the texture and edge feature distance measurements.

$$NORMAL\_LBP\_SM(i) = \frac{LBP\_SM(i) - \min(LBP\_SM)}{\max(LBP\_SM) - \min(LBP\_SM)}, i = 1, 2, \dots, K \quad (12)$$

$$NORMAL\_EDGE\_SM(i) = \frac{EDGE\_SM(i) - \min(EDGE\_SM)}{\max(EDGE\_SM) - \min(EDGE\_SM)}, i = 1, 2, \dots, K \quad (13)$$

Where K is the total number of images in the new dynamic database (which will change according to the selection rule in the first stage); LBP_SM(i) represents the similarity measure value based on LBP of the i-th image in the new database; min(LBP_SM) and max(LBP_SM) indicate the minimum and maximum texture feature similarity values for the entire set of images in the new database.

I. Displaying the product catalog search results.

The results from CTEBIR are presented as a product catalog for the user, complete with product names, prices, and product details. These products can then be ordered and purchased by the user.

J. Displaying the product catalog search results.

The visual search results will be evaluated using precision and recall calculations using Equations (14) and (15). High precision and recall scores indicate that the search results have a high level of accuracy and relevance.

$$Precission = \frac{Relevant\ Retrived}{All\ Relevant} \quad (14)$$

$$Recall = \frac{Relevant\ Retieved}{All\ Retrived} \quad (15)$$

Where:
True Positives (TP) are the correctly retrieved relevant items.
False Positives (FP) are the irrelevant items retrieved as relevant.
False Negatives (FN) are the relevant items not retrieved

III. RESULT AND DISCUSSION

The algorithm testing is conducted on two algorithms: the YOLO object detection algorithm and the CTEBIR algorithm for image search. The specifications of the devices used for algorithm testing can be seen in Table 2.

Table 2. Specifications of the Algorithm Testing Device

No	Specification	Server 1	Server 2	Client
1	CPU	Intel(R) Core(TM) i5-2400 3.10GHz	Intel® Celeron® B830 - 1.8GHz (2 Cores)	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
2	GPU	Nvidia GTX 460 with Max-Q Design (4 GB DDR 6 VRAM)	Intel HD Graphics	Nvidia GTX 1650 with Max-Q Design (4 GB DDR 6 VRAM)
3	RAM	8GB DDR3	8GB DDR3 1333	8 GB DDR 4
4	Storage	HDD 500GB	250GB (SSD - OS) 320GB (HDD)	256 GB 2666 MHz
5	Operation Systems	MX Linux XFCE 19.4 x64	Manjaro KDE 21.0.7	Windows 10 Home 64 Bit Version 20H2
6	Browser	Google Chrome v91.0.4472.124	Microsoft Edge Beta 93.0.961.11-1	Google Chrome v91.0.4472.124

A. Yolo Algorithm Testing

Manual and automatic testing of the YOLO algorithm are conducted by calculating the mAP from the detection output to assess its accuracy. The testing data consists of 60 testing samples from Kaggle for manual testing and 1,248 data samples from OpenImagev6 for automatic testing. An example of detection using YOLO can be seen in Figure 10 below.

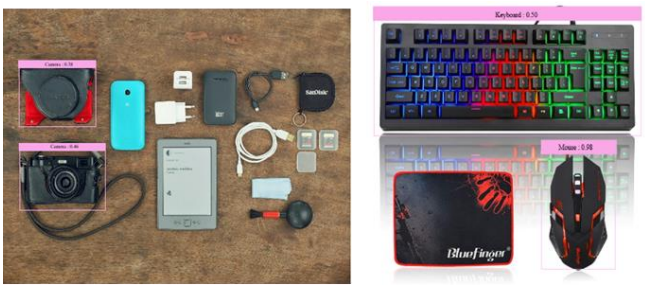

Figure 10. Detection Results of YOLOv4

The precision values from the manual testing detection results can be seen in Table 3.

Table 3. Table of Precision Values from Manual Testing Results using YOLOv4

No	Precision		
	Camera	Keyboard	Laptop
1	Camera: 46% Camera: 38%	Undetected	Laptop: 85%
2	Camera: 78%	Keyboard:86%	Laptop: 46%
3	Camera: 99%	Keyboard: 51%	Laptop: 65%
4	Camera: 94%	Keyboard: 32%	Laptop: 94%
5	Camera: 91%	Keyboard: 86%	Laptop: 88%
6	Camera: 95% Camera: 27%	Keyboard: 67%	Laptop: 67%
7	Camera: 93% Camera: 85% Camera: 28%	Keyboard: 55%	Laptop: 55%
8	Undetected	Keyboard: 67%	Laptop: 67%
9	Camera: 89%	Keyboard: 45%	Laptop: 45%
10	Camera: 71%	Keyboard: 33%	Laptop: 33%
Nos	**Precision**		
	Mouse	**Printer**	**Tablet**
1	Mouse: 54%	Printer: 54%	Tablet: 71%
2	Mouse: 90%	Printer: 27%	Tablet: 21%
3	Mouse: 71%	Printer: 28%	False Detection
4	Mouse: 97%	False Detection	False Detection
5	Mouse: 76%	Printer: 37%	Tablet: 47%
6	Mouse: 73%	Printer: 51%	Tablet: 85%
7	Mouse: 61%	Printer: 53%	Tablet: 65%
8	Mouse: 93%	Printer: 68%	False Detection
9	Mouse: 44%	Printer: 35%	Tablet: 60%
10	Mouse: 98%	Printer: 71%	Tablet: 26%

The testing results presented only include correct detections according to their categories. For detections that are incorrect or missed, a precision value of 0% is assigned. Out of 60 trials, 5 objects were incorrectly detected, and 2 inputs were missed by YOLOv4. Subsequently, the mAP is calculated by averaging the precision results from each category to obtain 6 APs (Average Precisions). These six APs are then averaged to yield the mAP. Based on the mAP calculation results in Table 4, it can be observed that the mAP value is 58.3375%. The process of calculating mAP can be seen in Table 4.

Table 4.Calculation Process of mAP for YOLOv4 Based on Table 3

No	Precision (%)					
	Camera	Keyboard	Laptop	Mouse	Printer	Tablet
1	42	0	85	54	54	71
2	78	86	46	90	27	27
3	99	51	65	71	28	0
4	94	32	94	97	0	0
5	91	86	88	76	37	47
6	95	67	84	73	51	85
7	58.25	55	0	61	53	65
8	0	67	84	93	68	0
9	89	45	67	44	35	60
10	71	33	86	98	71	26
AP (%)	71.725	52.2	69.9	75.7	42.4	38.1
					mAP (%) =	58.3375

Automatic testing on the YOLOv4 model yielded an mAP of 78.64% with a total detection time of 39 seconds, as shown in Figure 11 below. There is a difference of 20.31% between manual and automatic testing.

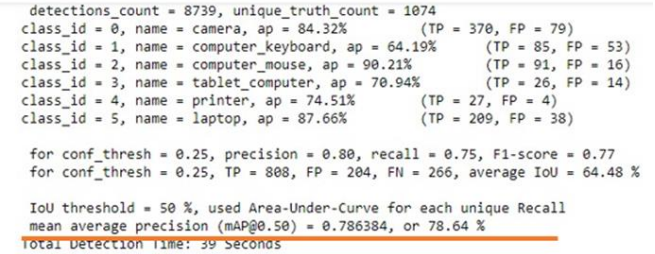

Figure 11. Results of YOLOv4 Testing on Validation Data

B. Testing the CTEBIR Algorithm

The CTEBIR algorithm is tested by calculating the Recall and Precision values for each category. The testing dataset consists of 60 data points.

Figure 12. Search Image: Before Cropping (Left) and After Cropping (Right)

The input data for this algorithm is obtained from the cropped images based on the bounding boxes detected by the YOLOv4 model in the previous stage. An example of the automatic cropping result based on the user-selected bounding box can be seen in Figure 12.

The search results are considered relevant if the colors match, then further evaluated based on shape and type. For the camera category, images in the database are divided into 3 types of cameras: digital camera, mirrorless, and DSLR. The query image (input) is black in color and of mirrorless type. Therefore, the relevant retrieved images are 8, the total retrieved images are 9, and the total relevant images in the database are 11. The calculation of Recall and Precision can be obtained using Equations (16) and (17).

$$Recall = \frac{relevant\ retrieved}{all\ relevant}\ x\ 100\% = \frac{8}{11} x\ 100\% = 72.73\ \% \quad (16)$$

$$Precision = \frac{relevant\ retrieved}{all\ retrieved}\ x\ 100\% = \frac{8}{9} x\ 100\% = 88.89\ \% \quad (17)$$

Calculation of the average recall and precision results from the overall testing can be seen in Table 5 below.

Table 5. Overall Results of CTEBIR Algorithm Testing

No	Category	Precision	Recall	Time Research (s)
1	Camera	62.67%	46.39%	19.015
2	Keyboard	46.88%	47.08%	22.532
3	Laptop	52.50%	63.69%	65.332
4	Mouse	40.20%	56.70%	40.929
5	Printer	38.00%	32.74%	84.911
6	Tablet	35.17%	19.14%	56.219
Mean		45.90%	42.29%	43.2

From the above test results, it can be seen that the average recall value is 44.29%, the average precision value is 45.90%, and the average search time is 43.2 seconds.

Figure 13. Results of the CTEBIR Search

These three values are highly dependent on the detection results in the previous stage. If the object detection results in the previous stage provide incorrect categories/names, then the search results will also be incorrect automatically. Meanwhile, if the object detection results do not produce any classes, then the search cannot be conducted. Examples of test results in the form of images can be seen in Figure 13.

C. Testing the CTEBIR Algorithm

The CTEBIR algorithm was also tested using two servers with different hardware specifications, aiming to determine the impact of hardware specifications differences on performance. The testing data consists of two images from each category sourced from the previous 60 testing data. Explanation of the testing data can be found in Table 6 below.

Table 6. Testing Data: Two Images per Category from Previous 60 Datasets

Category	Image	Relevant Retrived	All Retrived	All Relevant
Camera	1	1	1	1
	2	6	9	11
Keyboard	1	6	8	10
	2	4	7	7
Laptop	1	4	10	11
	2	3	6	13
Mouse	1	3	10	5
	2	8	10	20
Printer	1	8	10	12
	2	5	10	9
Tablet	1	0	2	12
	2	4	6	14

Table 7. Comparison of Search Time Between Server 1 and Server 2 Running CTEBIR Algorithm

Category	Precision	Recall	Server 1 Retrieval Time (s)	Server 2 Retrieval Time (s)
Camera	100.00%	100.00%	8.66	24.00
	66.67%	54.55%	24.61	60.00
Keyboard	75.00%	60.00%	26.68	57.00
	57.14%	57.14%	17.63	40.00
Laptop	40.00%	90.91%	22.24	49.00
	50.00%	23.08%	17.13	33.50
Mouse	30.0%	60.0%	27.52	60.80
	80.0%	40.0%	52.33	116.60
Printer	80.00%	66.67%	122.91	278.70
	50.00%	55.56%	33.40	60.00
Tablet	0.00%	0.00%	8.22	50.00
	66.67%	28.57%	18.08	40.20
Average	57.96%	53.04%	31.62	72.48

From the Table 7, it can be observed that the search process when the CTEBIR algorithm is run on server 2 takes approximately twice as long as on server 1, where the search time using server 1 is 31.62 seconds and the search time using server 2 is 72.48 seconds. This proves that the search time depends on the device specifications.

IV. CONCLUSION

Based on the conducted testing, the following conclusions can be drawn:

1. Our experiments show that utilizing the YOLOv4 algorithm can enhance the accuracy and speed of visual searches by simplifying the search process based on

identified classes. Additionally, our precision assessment yielded a score of 95%, with individual scores for camera objects reaching 90%, keyboards achieving 85%, and laptops attaining 71%. These findings bolster the reliability of the CTEBIR algorithm in image matching and provide a deeper understanding of the system's effectiveness in accurately detecting and distinguishing objects.

2. Recall, precision, and search time with the CTEBIR algorithm heavily depend on the object detection results obtained by the YOLOv4 algorithm. If the object detection produces incorrect classes, then the CTEBIR search results will inevitably be irrelevant.
3. Implementing CTEBIR in object detection using the YOLO algorithm can enhance the accuracy and speed of the image search process by directly influencing the search space for these images, enabling their detection according to their respective classes.

REFERENCES

[1] S. Amin and K. Kansana, "A Review Paper on E-Commerce," 2016. [Online]. Available: https://www.researchgate.net/publication/304703920

[2] G. Anand, S. Wang, and K. Ni, "Large-scale visual search and similarity for e-commerce," in *Applications of Machine Learning 2021*, M. E. Zelinski, T. M. Taha, and J. Howe, Eds., SPIE, Aug. 2021, p. 31. doi: 10.1117/12.2594924.

[3] M. P. Eckstein, "Visual search: A retrospective," *J Vis*, vol. 11, no. 5, pp. 14–14, Dec. 2011, doi: 10.1167/11.5.14.

[4] S. Vijayanarasimhan and K. Grauman, "Efficient region search for object detection," in *CVPR 2011*, IEEE, Jun. 2011, pp. 1401–1408. doi: 10.1109/CVPR.2011.5995545.

[5] V. Narayanan and M. Likhachev, "PERCH: Perception via search for multi-object recognition and localization," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2016, pp. 5052–5059. doi: 10.1109/ICRA.2016.7487711.

[6] Y. F. Dewi and N. Fadillah, "Deteksi Objek Berwarna Merah Secara Real Time Dengan Algoritma Color Filtering," *Jurnal Media Informatika Budidarma*, vol. 3, no. 2, p. 140, Apr. 2019, doi: 10.30865/mib.v3i2.1114.

[7] Q. Liu, X. Fan, Z. Xi, Z. Yin, and Z. Yang, "Object detection based on Yolov4-Tiny and Improved Bidirectional feature pyramid network," *J Phys Conf Ser*, vol. 2209, no. 1, p. 012023, Feb. 2022, doi: 10.1088/1742-6596/2209/1/012023.

[8] E. R. Setyaningsih and M. S. Edy, "YOLOv4 dan Mask R-CNN Untuk Deteksi Kerusakan Pada Karung Komoditi," *Teknika*, vol. 11, no. 1, pp. 45–52, Mar. 2022, doi: 10.34148/teknika.v11i1.419.

[9] R. Gai, N. Chen, and H. Yuan, "A detection algorithm for cherry fruits based on the improved YOLO-v4 model," *Neural Comput Appl*, vol. 35, no. 19, pp. 13895–13906, Jul. 2023, doi: 10.1007/s00521-021-06029-z.

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.10934

[11] J. Yu and W. Zhang, "Face Mask Wearing Detection Algorithm Based on Improved YOLO-v4," *Sensors*, vol. 21, no. 9, p. 3263, May 2021, doi: 10.3390/s21093263.

[12] L. K. Pavithra and T. S. Sharmila, "An efficient framework for image retrieval using color, texture and edge features," *Computers & Electrical Engineering*, vol. 70, pp. 580–593, Aug. 2018, doi: 10.1016/j.compeleceng.2017.08.030.

[13] G. Ramaditya and W. F. Al Maki, "Single Object Tracking with Minimum False Positive using YOLOv4, VGG16, and Cosine Distance," *Jurnal Media Informatika Budidarma*, vol. 6, no. 4, p. 2196, Oct. 2022, doi: 10.30865/mib.v6i4.4827.

[14] N. D. Lynn, A. I. Sourav, and A. J. Santoso, "Implementation of Real-Time Edge Detection Using Canny and Sobel Algorithms," *IOP Conf Ser Mater Sci Eng*, vol. 1096, no. 1, p. 012079, Mar. 2021, doi: 10.1088/1757-899X/1096/1/012079.

[15] X. Zhou, L. Jiang, C. Hu, S. Lei, T. Zhang, and X. Mou, "YOLO-SASE: An Improved YOLO Algorithm for the Small Targets Detection in Complex Backgrounds," *Sensors*, vol. 22, no. 12, p. 4600, Jun. 2022, doi: 10.3390/s22124600.

[16] P. Laia, "Penerapan Metode Prewitt, Canny dan Sobel Pada Proses Deteksi Tepi Citra," *Jurnal Media Informatika Budidarma*, vol. 2, no. 1, Jan. 2018, doi: 10.30865/mib.v2i1.996.

.